

```
1
2 #! /usr/bin/env python3
3
4
5 ### Multi-Step Tactile Sensor Reading Test Script ###
6
7 ###
8 # KINOVA (R) KORTEX (TM)
9 #
10 # Copyright (c) 2018 Kinova inc. All rights reserved.
11 #
12 # This software may be modified and distributed
13 # under the terms of the BSD 3-Clause license.
14 #
15 # Refer to the LICENSE file for details.
16 #
17 ###
18
19 from sunau import AUDIO_UNKNOWN_SIZE
20 import sys
21 import os
22 import time
23 import threading
24 import numbers
25
26 from kortex_api.autogen.client_stubs.BaseClientRpc import BaseClient
27 from kortex_api.autogen.client_stubs.BaseCyclicClientRpc import BaseCyclicClient
28
29 from kortex_api.autogen.messages import Base_pb2, BaseCyclic_pb2, Common_pb2
30
31 # DEFINITIONS:
32
33 # Index Variables
34 # Initialize the variables for the x,y and z slots of the soft and hard
35 # list.
36 x = 0;
37 y = 1;
38 z = 2;
39
40 # Sensor Dimensions - Unit: cm
41 height = 11.5;
42 diameter = 6.5;
43 radius = diameter/2;
44
45
46 # Soft Inclusion Phantom Center Coordinates - Unit: cm
47 soft = [45, 5, -1.5];
48
49 # Subtracting the radius of the sample tube so the center of the
50 # sample tube sits above the center of the phantom.
51 softx = soft[x] - radius
52 softy = soft[y]
53
54 # Subtracting the height of the sample tube so the edge of the
55 # sample tube hovers above the phantom.
56 softz = soft[z] + height
57 print(softx, softy, softz)
58
59 # Hard Inclusion Phantom Center Coordinates - Unit: cm
```

```

60 hard = [45, -5, -0.5];
61 hardx = hard[x] - radius
62 hardy = hard[y]
63 hardz = hard[z] + height
64 print(hardx, hardy, hardz)
65
66 # End Effector Angle - Unit: degrees
67 # These angles are chosen to fix the end-effector of the Kinova arm at
68 # 90-degree angle.
69 adx = 90;
70 ady = 0;
71 adz = 90;
72
73 # Pyrex Coordinates - Unit: cm
74 P1 = [47, 10, -3.5]; #Top Left
75 P2 = [47, 10, -3.5]; #Top Right
76 P3 = [30, -14, -3.5]; #Bottom Right
77 P4 = [30, 10, -3.5]; #Bottom Left
78
79 # Maximum allowed waiting time during actions (in seconds)
80 TIMEOUT_DURATION = 1000
81
82 # Create closure to set an event after an END or an ABORT
83 # Takes an input of "e" to see if the program has finished or aborted.
84 # Returns the notification of the event that occurred.
85
86 def check_for_end_or_abort(e):
87     """Return a closure checking for END or ABORT notifications
88     Arguments:
89     e -- event to signal when the action is completed
90         (will be set when an END or ABORT occurs)
91     """
92     def check(notification, e = e):
93         print("EVENT : " + \
94             Base_pb2.ActionEvent.Name(notification.action_event))
95         if notification.action_event == Base_pb2.ACTION_END \
96         or notification.action_event == Base_pb2.ACTION_ABORT:
97             e.set()
98     return check
99
100 def example_test_movement(base, base_cyclic):
101
102     print("Starting Cartesian action movement ...")
103     action = Base_pb2.Action()
104     action.name = "Example Cartesian action movement"
105     action.application_data = ""
106
107     feedback = base_cyclic.RefreshFeedback()
108
109     cartesian_pose = action.reach_pose.target_pose
110     cartesian_pose.x = feedback.base.tool_pose_x # (meters)
111     cartesian_pose.y = feedback.base.tool_pose_y # (meters)
112     cartesian_pose.z = feedback.base.tool_pose_z # (meters)
113     cartesian_pose.theta_x = feedback.base.tool_pose_theta_x # (degrees)
114     cartesian_pose.theta_y = feedback.base.tool_pose_theta_y # (degrees)
115     cartesian_pose.theta_z = feedback.base.tool_pose_theta_z # (degrees)
116
117     e = threading.Event()
118     notification_handle = base.OnNotificationActionTopic(
119         check_for_end_or_abort(e),

```

```
120     Base_pb2.NotificationOptions()
121 )
122
123 print("Executing action")
124 base.ExecuteAction(action)
125
126 print("Waiting for movement to finish ...")
127 finished = e.wait(TIMEOUT_DURATION)
128 base.Unsubscribe(notification_handle)
129
130
131 # Soft Movement 1
132 print("Starting Cartesian action movement ...")
133 action1 = Base_pb2.Action()
134 action1.name = "Movement 1- Top Left"
135 action1.application_data = ""
136
137 feedback = base_cyclic.RefreshFeedback()
138
139 cartesian_pose = action1.reach_pose.target_pose
140 # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
141
142 cartesian_pose.x = softx/100          # (meters)
143 cartesian_pose.y = softy/100          # (meters)
144 cartesian_pose.z = softz/100 + 15/100 # (meters)
145 cartesian_pose.theta_x = adx          # (degrees)
146 cartesian_pose.theta_y = ady          # (degrees)
147 cartesian_pose.theta_z = adz          # (degrees)
148
149 e = threading.Event()
150 notification_handle = base.OnNotificationActionTopic(
151     check_for_end_or_abort(e),
152     Base_pb2.NotificationOptions()
153 )
154
155 print("Executing action")
156 base.ExecuteAction(action1)
157
158 print("Waiting for movement to finish ...")
159 finished = e.wait(TIMEOUT_DURATION)
160 base.Unsubscribe(notification_handle)
161
162 # Soft Movement 2
163 print("Starting Cartesian action movement ...")
164 action1 = Base_pb2.Action()
165 action1.name = "Movement 1- Top Left"
166 action1.application_data = ""
167
168 feedback = base_cyclic.RefreshFeedback()
169
170 cartesian_pose = action1.reach_pose.target_pose
171 # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
172
173 cartesian_pose.x = softx/100          # (meters)
174 cartesian_pose.y = softy/100          # (meters)
175 cartesian_pose.z = softz/100          # (meters)
176 cartesian_pose.theta_x = adx          # (degrees)
177 cartesian_pose.theta_y = ady          # (degrees)
```

```
178     cartesian_pose.theta_z = adz      # (degrees)
179
180     e = threading.Event()
181     notification_handle = base.OnNotificationActionTopic(
182         check_for_end_or_abort(e),
183         Base_pb2.NotificationOptions()
184     )
185
186     print("Executing action")
187     base.ExecuteAction(action1)
188
189     print("Waiting for movement to finish ...")
190     finished = e.wait(TIMEOUT_DURATION)
191     base.Unsubscribe(notification_handle)
192
193 # Soft Movement 3
194     print("Starting Cartesian action movement ...")
195     action1 = Base_pb2.Action()
196     action1.name = "Movement 1- Top Left"
197     action1.application_data = ""
198
199     feedback = base_cyclic.RefreshFeedback()
200
201     cartesian_pose = action1.reach_pose.target_pose
202     # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
203
204     cartesian_pose.x = softx/100      # (meters)
205     cartesian_pose.y = softy/100      # (meters)
206     cartesian_pose.z = softz/100 - 0.2/100      # (meters)
207     cartesian_pose.theta_x = adx      # (degrees)
208     cartesian_pose.theta_y = ady      # (degrees)
209     cartesian_pose.theta_z = adz      # (degrees)
210
211     e = threading.Event()
212     notification_handle = base.OnNotificationActionTopic(
213         check_for_end_or_abort(e),
214         Base_pb2.NotificationOptions()
215     )
216
217     print("Executing action")
218     base.ExecuteAction(action1)
219
220     print("Waiting for movement to finish ...")
221     finished = e.wait(TIMEOUT_DURATION)
222     base.Unsubscribe(notification_handle)
223
224 # Soft Movement 4
225     print("Starting Cartesian action movement ...")
226     action1 = Base_pb2.Action()
227     action1.name = "Movement 1- Top Left"
228     action1.application_data = ""
229
230     feedback = base_cyclic.RefreshFeedback()
231
232     cartesian_pose = action1.reach_pose.target_pose
233     # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
234
235     cartesian_pose.x = softx/100      # (meters)
```

```
236     cartesian_pose.y = softy/100          # (meters)
237     cartesian_pose.z = softz/100 - 0.4/100 # (meters)
238     cartesian_pose.theta_x = adx          # (degrees)
239     cartesian_pose.theta_y = ady          # (degrees)
240     cartesian_pose.theta_z = adz          # (degrees)
241
242     e = threading.Event()
243     notification_handle = base.OnNotificationActionTopic(
244         check_for_end_or_abort(e),
245         Base_pb2.NotificationOptions()
246     )
247
248     print("Executing action")
249     base.ExecuteAction(action1)
250
251     print("Waiting for movement to finish ...")
252     finished = e.wait(TIMEOUT_DURATION)
253     base.Unsubscribe(notification_handle)
254
255 # Soft Movement 5
256     print("Starting Cartesian action movement ...")
257     action1 = Base_pb2.Action()
258     action1.name = "Movement 1- Top Left"
259     action1.application_data = ""
260
261     feedback = base_cyclic.RefreshFeedback()
262
263     cartesian_pose = action1.reach_pose.target_pose
264     # These are loaded directly from Kinova Web app (the first three are divided by
265     100 since it gives them in mm)
266
267     cartesian_pose.x = softx/100          # (meters)
268     cartesian_pose.y = softy/100          # (meters)
269     cartesian_pose.z = softz/100 - 0.6/100 # (meters)
270     cartesian_pose.theta_x = adx          # (degrees)
271     cartesian_pose.theta_y = ady          # (degrees)
272     cartesian_pose.theta_z = adz          # (degrees)
273
274     e = threading.Event()
275     notification_handle = base.OnNotificationActionTopic(
276         check_for_end_or_abort(e),
277         Base_pb2.NotificationOptions()
278     )
279
280     print("Executing action")
281     base.ExecuteAction(action1)
282
283     print("Waiting for movement to finish ...")
284     finished = e.wait(TIMEOUT_DURATION)
285     base.Unsubscribe(notification_handle)
286
287 # Soft Movement 6
288     print("Starting Cartesian action movement ...")
289     action1 = Base_pb2.Action()
290     action1.name = "Movement 1- Top Left"
291     action1.application_data = ""
292
293     feedback = base_cyclic.RefreshFeedback()
294
295     cartesian_pose = action1.reach_pose.target_pose
```

```
295 # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
296
297 cartesian_pose.x = softx/100 # (meters)
298 cartesian_pose.y = softy/100 # (meters)
299 cartesian_pose.z = softz/100 - 0.8/100 # (meters)
300 cartesian_pose.theta_x = adx # (degrees)
301 cartesian_pose.theta_y = ady # (degrees)
302 cartesian_pose.theta_z = adz # (degrees)
303
304 e = threading.Event()
305 notification_handle = base.OnNotificationActionTopic(
306     check_for_end_or_abort(e),
307     Base_pb2.NotificationOptions()
308 )
309
310 print("Executing action")
311 base.ExecuteAction(action1)
312
313 print("Waiting for movement to finish ...")
314 finished = e.wait(TIMEOUT_DURATION)
315 base.Unsubscribe(notification_handle)
316
317 # Soft Movement 7
318 print("Starting Cartesian action movement ...")
319 action1 = Base_pb2.Action()
320 action1.name = "Movement 1- Top Left"
321 action1.application_data = ""
322
323 feedback = base_cyclic.RefreshFeedback()
324
325 cartesian_pose = action1.reach_pose.target_pose
326 # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
327
328 cartesian_pose.x = softx/100 # (meters)
329 cartesian_pose.y = softy/100 # (meters)
330 cartesian_pose.z = softz/100 - 2/100 # (meters)
331 cartesian_pose.theta_x = adx # (degrees)
332 cartesian_pose.theta_y = ady # (degrees)
333 cartesian_pose.theta_z = adz # (degrees)
334
335 e = threading.Event()
336 notification_handle = base.OnNotificationActionTopic(
337     check_for_end_or_abort(e),
338     Base_pb2.NotificationOptions()
339 )
340
341 print("Executing action")
342 base.ExecuteAction(action1)
343
344 print("Waiting for movement to finish ...")
345 finished = e.wait(TIMEOUT_DURATION)
346 base.Unsubscribe(notification_handle)
347
348 # Retracting Arm
349 print("Starting Cartesian action movement ...")
350 action1 = Base_pb2.Action()
351 action1.name = "Movement 1- Top Left"
352 action1.application_data = ""
```

```
353
354     feedback = base_cyclic.RefreshFeedback()
355
356     cartesian_pose = action1.reach_pose.target_pose
357     # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
358
359     cartesian_pose.x = softx/100          # (meters)
360     cartesian_pose.y = softy/100          # (meters)
361     cartesian_pose.z = softz/100 + 15/100 # (meters)
362     cartesian_pose.theta_x = adx          # (degrees)
363     cartesian_pose.theta_y = ady          # (degrees)
364     cartesian_pose.theta_z = adz          # (degrees)
365
366     e = threading.Event()
367     notification_handle = base.OnNotificationActionTopic(
368         check_for_end_or_abort(e),
369         Base_pb2.NotificationOptions()
370     )
371
372     print("Executing action")
373     base.ExecuteAction(action1)
374
375     print("Waiting for movement to finish ...")
376     finished = e.wait(TIMEOUT_DURATION)
377     base.Unsubscribe(notification_handle)
378
379 # Transition to Hard Inclusion
380     print("Starting Cartesian action movement ...")
381     action1 = Base_pb2.Action()
382     action1.name = "Movement 1- Top Left"
383     action1.application_data = ""
384
385     feedback = base_cyclic.RefreshFeedback()
386
387     cartesian_pose = action1.reach_pose.target_pose
388     # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
389
390     cartesian_pose.x = hardx/100          # (meters)
391     cartesian_pose.y = hardy/100          # (meters)
392     cartesian_pose.z = hardz/100 + 15/100 # (meters)
393     cartesian_pose.theta_x = adx          # (degrees)
394     cartesian_pose.theta_y = ady          # (degrees)
395     cartesian_pose.theta_z = adz          # (degrees)
396
397     e = threading.Event()
398     notification_handle = base.OnNotificationActionTopic(
399         check_for_end_or_abort(e),
400         Base_pb2.NotificationOptions()
401     )
402
403     print("Executing action")
404     base.ExecuteAction(action1)
405
406     print("Waiting for movement to finish ...")
407     finished = e.wait(TIMEOUT_DURATION)
408     base.Unsubscribe(notification_handle)
409
410 # Hard Movement 1
```



```

411     print("Starting Cartesian action movement ...")
412     action1 = Base_pb2.Action()
413     action1.name = "Movement 1- Top Left"
414     action1.application_data = ""
415
416     feedback = base_cyclic.RefreshFeedback()
417
418     cartesian_pose = action1.reach_pose.target_pose
419     # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
420
421     cartesian_pose.x = hardx/100          # (meters)
422     cartesian_pose.y = hardy/100          # (meters)
423     cartesian_pose.z = hardz/100          # (meters)
424     cartesian_pose.theta_x = adx          # (degrees)
425     cartesian_pose.theta_y = ady          # (degrees)
426     cartesian_pose.theta_z = adz          # (degrees)
427
428     e = threading.Event()
429     notification_handle = base.OnNotificationActionTopic(
430         check_for_end_or_abort(e),
431         Base_pb2.NotificationOptions()
432     )
433
434     print("Executing action")
435     base.ExecuteAction(action1)
436
437     print("Waiting for movement to finish ...")
438     finished = e.wait(TIMEOUT_DURATION)
439     base.Unsubscribe(notification_handle)
440
441 # Hard Movement 2
442     print("Starting Cartesian action movement ...")
443     action1 = Base_pb2.Action()
444     action1.name = "Movement 1- Top Left"
445     action1.application_data = ""
446
447     feedback = base_cyclic.RefreshFeedback()
448
449     cartesian_pose = action1.reach_pose.target_pose
450     # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
451
452     cartesian_pose.x = hardx/100          # (meters)
453     cartesian_pose.y = hardy/100          # (meters)
454     cartesian_pose.z = hardz/100 - 0.2/100 # (meters)
455     cartesian_pose.theta_x = adx          # (degrees)
456     cartesian_pose.theta_y = ady          # (degrees)
457     cartesian_pose.theta_z = adz          # (degrees)
458
459     e = threading.Event()
460     notification_handle = base.OnNotificationActionTopic(
461         check_for_end_or_abort(e),
462         Base_pb2.NotificationOptions()
463     )
464
465     print("Executing action")
466     base.ExecuteAction(action1)
467
468     print("Waiting for movement to finish ...")

```



```
469     finished = e.wait(TIMEOUT_DURATION)
470     base.Unsubscribe(notification_handle)
471
472 # Soft Movement 3
473     print("Starting Cartesian action movement ...")
474     action1 = Base_pb2.Action()
475     action1.name = "Movement 1- Top Left"
476     action1.application_data = ""
477
478     feedback = base_cyclic.RefreshFeedback()
479
480     cartesian_pose = action1.reach_pose.target_pose
481     # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
482
483     cartesian_pose.x = hardx/100          # (meters)
484     cartesian_pose.y = hardy/100          # (meters)
485     cartesian_pose.z = hardz/100 - 0.4/100 # (meters)
486     cartesian_pose.theta_x = adx          # (degrees)
487     cartesian_pose.theta_y = ady          # (degrees)
488     cartesian_pose.theta_z = adz          # (degrees)
489
490     e = threading.Event()
491     notification_handle = base.OnNotificationActionTopic(
492         check_for_end_or_abort(e),
493         Base_pb2.NotificationOptions()
494     )
495
496     print("Executing action")
497     base.ExecuteAction(action1)
498
499     print("Waiting for movement to finish ...")
500     finished = e.wait(TIMEOUT_DURATION)
501     base.Unsubscribe(notification_handle)
502
503 # Hard Movement 4
504     print("Starting Cartesian action movement ...")
505     action1 = Base_pb2.Action()
506     action1.name = "Movement 1- Top Left"
507     action1.application_data = ""
508
509     feedback = base_cyclic.RefreshFeedback()
510
511     cartesian_pose = action1.reach_pose.target_pose
512     # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
513
514     cartesian_pose.x = hardx/100          # (meters)
515     cartesian_pose.y = hardy/100          # (meters)
516     cartesian_pose.z = hardz/100 - 0.6/100 # (meters)
517     cartesian_pose.theta_x = adx          # (degrees)
518     cartesian_pose.theta_y = ady          # (degrees)
519     cartesian_pose.theta_z = adz          # (degrees)
520
521     e = threading.Event()
522     notification_handle = base.OnNotificationActionTopic(
523         check_for_end_or_abort(e),
524         Base_pb2.NotificationOptions()
525     )
526
```

```
527     print("Executing action")
528     base.ExecuteAction(action1)
529
530     print("Waiting for movement to finish ...")
531     finished = e.wait(TIMEOUT_DURATION)
532     base.Unsubscribe(notification_handle)
533
534 # Hard Movement 5
535     print("Starting Cartesian action movement ...")
536     action1 = Base_pb2.Action()
537     action1.name = "Movement 1- Top Left"
538     action1.application_data = ""
539
540     feedback = base_cyclic.RefreshFeedback()
541
542     cartesian_pose = action1.reach_pose.target_pose
543     # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
544
545     cartesian_pose.x = hardx/100          # (meters)
546     cartesian_pose.y = hardy/100          # (meters)
547     cartesian_pose.z = hardz/100 - 0.8/100 # (meters)
548     cartesian_pose.theta_x = adx          # (degrees)
549     cartesian_pose.theta_y = ady          # (degrees)
550     cartesian_pose.theta_z = adz          # (degrees)
551
552     e = threading.Event()
553     notification_handle = base.OnNotificationActionTopic(
554         check_for_end_or_abort(e),
555         Base_pb2.NotificationOptions()
556     )
557
558     print("Executing action")
559     base.ExecuteAction(action1)
560
561     print("Waiting for movement to finish ...")
562     finished = e.wait(TIMEOUT_DURATION)
563     base.Unsubscribe(notification_handle)
564
565 # hard Movement 6
566     print("Starting Cartesian action movement ...")
567     action1 = Base_pb2.Action()
568     action1.name = "Movement 1- Top Left"
569     action1.application_data = ""
570
571     feedback = base_cyclic.RefreshFeedback()
572
573     cartesian_pose = action1.reach_pose.target_pose
574     # These are loaded directly from Kinova Web app (the first three are divided by
100 since it gives them in mm)
575
576     cartesian_pose.x = hardx/100          # (meters)
577     cartesian_pose.y = hardy/100          # (meters)
578     cartesian_pose.z = hardz/100 - 2/100  # (meters)
579     cartesian_pose.theta_x = adx          # (degrees)
580     cartesian_pose.theta_y = ady          # (degrees)
581     cartesian_pose.theta_z = adz          # (degrees)
582
583     e = threading.Event()
584     notification_handle = base.OnNotificationActionTopic(
```

```

585     check_for_end_or_abort(e),
586     Base_pb2.NotificationOptions()
587 )
588
589     print("Executing action")
590     base.ExecuteAction(action1)
591
592     print("Waiting for movement to finish ...")
593     finished = e.wait(TIMEOUT_DURATION)
594     base.Unsubscribe(notification_handle)
595
596 # Retracting Arm
597     print("Starting Cartesian action movement ...")
598     action1 = Base_pb2.Action()
599     action1.name = "Movement 1- Top Left"
600     action1.application_data = ""
601
602     feedback = base_cyclic.RefreshFeedback()
603
604     cartesian_pose = action1.reach_pose.target_pose
605     # These are loaded directly from Kinova Web app (the first three are divided by
606     100 since it gives them in mm)
607     cartesian_pose.x = hardx/100          # (meters)
608     cartesian_pose.y = hardy/100          # (meters)
609     cartesian_pose.z = hardz/100 + 15/100 # (meters)
610     cartesian_pose.theta_x = adx          # (degrees)
611     cartesian_pose.theta_y = ady          # (degrees)
612     cartesian_pose.theta_z = adz          # (degrees)
613
614     e = threading.Event()
615     notification_handle = base.OnNotificationActionTopic(
616         check_for_end_or_abort(e),
617         Base_pb2.NotificationOptions()
618     )
619
620     print("Executing action")
621     base.ExecuteAction(action1)
622
623     print("Waiting for movement to finish ...")
624     finished = e.wait(TIMEOUT_DURATION)
625     base.Unsubscribe(notification_handle)
626
627     if finished:
628         print("Cartesian movement completed")
629     else:
630         print("Timeout on action notification wait")
631     return finished
632
633 def main():
634
635     # Import the utilities helper module
636     sys.path.insert(0, os.path.join(os.path.dirname(__file__), ".."))
637     import utilities
638
639     # Parse arguments
640     args = utilities.parseConnectionArguments()
641
642     # Create connection to the device and get the router
643     with utilities.DeviceConnection.createTcpConnection(args) as router:

```

```
644
645     # Create required services
646     base = BaseClient(router)
647     base_cyclic = BaseCyclicClient(router)
648
649     # Example core
650     success = True
651     success &= example_test_movement(base, base_cyclic)
652     #success &= example_test_movement(base, base_cyclic)
653
654     return 0 if success else 1
655
656 if __name__ == "__main__":
657     exit(main())
658
```